Generalized Linear Model Investigation

S. NO	Module/ Framework/ Package	Name and a brief description of the algorithm	An example of a situation where using the provided GLM implementation provides superior performance compared to that of base R or its equivalent in Python (identify the equivalent in Python)
a.	Base R(stats package)	The Generalized Linear Model (GLM) algorithm in R is implemented using the glm() function. The glm()function primarily uses the **Iteratively Reweighted Least Squares (IRLS) algorithm** to iteratively estimate model parameters, making it efficient for small to medium-sized datasets.[1] By using the family argument, it supports a variety of distributions, such as Gaussian, Binomial, Poisson, Gamma, and Inverse Gaussian.[1]	**Use case:** Suppose we conduct a health survey with 5,000 participants, collecting data on age, gender, physical activity level (light, moderate, high), BMI, and dietary habits. Our goal is to predict whether an individual has a risk of developing heart disease (yes/no) using logistic regression (GLM with a binomial family). **Reasons:** glm() in R is optimized for small datasets and frequently outperforms Python's statsmodels.GLM(), which has additional computational overhead. Unlike Python's statsmodels.GLM(), which involves manual encoding of categorical variables (e.g., 'light', 'moderate', 'high' into dummy variables), R's glm() rapidly recognizes and processes categorical data, decreasing the number of preprocessing steps. Hence, Base R demonstrates greater performance in this scenario.
b.	Big data version of R	**H2O glm:** H2O glm is a scalable, parallelized implementation of GLM that enables distributed computing on big datasets. H2O GLM offers regularization (L1/L2), elastic nets, and other modifications to improve predictive performance.[2] **Biglm:** Biglm is a regression model for datasets that are too large to fit into memory. It supports incremental updates to the model using data chunks, making it ideal for dealing with large-scale regression problems without requiring the entire dataset to be loaded at once.[2]	**Use case:** A national healthcare system is analyzing 50 million electronic health records (EHRs) to predict which diabetes patients are at high risk of hospital readmission within 30 days. The dataset includes patient demographics, medical history, medication usage, lab results, and previous hospital visits. The goal is to fit a logistic regression model (GLM with binomial family) to identify risk factors and optimize hospital resource allocation. **Reasons:** Base R's glm() function involves loading the

			complete dataset into memory, which is difficult for such a large dataset. H2O glm and biglm process data in pieces or distribute calculations across numerous nodes, allowing them to scale. Biglm can incrementally update models as new patient records are received, making it ideal for real-time hospital monitoring. As a result, the Big data version of R is considered preferable in this scenario.
c.	Dask ML	**ADMM (Alternating Direction Method of Multipliers)**: A distributed optimization algorithm for solving GLMs with L1 or L2 regularization by splitting the problem into smaller subproblems, making it efficient for large datasets.[3] **Compute Step Size Dask**: Computes the optimal step size for gradient-based optimization methods using backtracking and Armijo rule to ensure convergence stability.[3] **Gradient Descent**: A first-order optimization algorithm that iteratively updates model parameters to minimize the loss function, using a fixed or adaptive learning rate.[3] **L-BFGS (Limited-memory Broyden-Fletcher-Goldfarb-Shanno Algorithm)**: A quasi-Newton optimization method that efficiently estimates gradients using a limited memory approach, improving performance on large datasets.[3] **Newton's Method**: A second-order optimization algorithm that uses Hessian information to update parameters more efficiently than first-order methods, particularly useful for logistic regression.[3] **Proximal Gradient Method**: A gradient-based optimization method that incorporates regularization (e.g., L1 or L2) using a proximal operator to handle sparsity in large-scale datasets.[3]	**Use case:** A hospital network is analyzing 100 million ICU patient records from multiple locations to predict sepsis risk. The dataset includes vital signs (heart rate, blood pressure, temperature), lab test results, medications, and medical history. The goal is to train a logistic regression model (GLM with binomial family) using large-scale distributed data while ensuring efficient computation. **Reasons:**Dask-ML GLM distributes data across numerous nodes, allowing for scalable processing. It can conduct computations across multiple CPU cores and distributed clusters. Base R's glm() only uses IRLS, which is not optimized for large-scale computing, whereas Dask-ML supports ADMM, L-BFGS, and Newton's Method, making it faster and more efficient than Python's statsmodels.GLM() for large datasets. As a result, DaskML outperforms others in this scenario.

d.	Spark R	Fits GLMs on large-scale data stored in a SparkDataFrame. Supports multiple distributions (e.g., Gaussian, Binomial, Poisson, Gamma, Tweedie) and allows regularization (L2). Implements an **Iteratively Reweighted Least Squares (IRLS)** approach for optimization and supports regularization, weight columns, and string encoding options. It involves solving a weighted least squares problem at each iteration.[4]	**Use case:** A nationwide hospital network wants to predict the number of emergency room (ER) visits per hospital per day to optimize staffing and resource allocation. The dataset consists of millions of hospital records containing Date & time of visits, Hospital location, Patient demographics (age, gender, medical history). Since the outcome is a count variable (ER visit volume per day), we use Poisson regression, a type of GLM suited for modeling count data. **Reasons:** Spark R uses Iteratively Reweighted Least Squares (IRLS) in a distributed setting, significantly improving computation speed but, Base R's glm() and Python's statsmodels.GLM() rely on single-node IRLS, leading to slower performance on large datasets. Spark R can process real-time ER visit data from hospital databases, making it ideal for dynamic staffing adjustments, whereas Base R and Python's statsmodels.GLM() require batch processing, limiting real-time decision-making. Hence, Spark R is considered superior to Base R/ Python's statsmodels.GLM() in this case.
e.	Spark Optimization	**Gradient Descent (GD) & Stochastic Gradient Descent (SGD)**: Iterative optimization methods used for minimizing an objective function by taking steps in the direction of the negative gradient. SGD randomly selects subsets of data to compute gradients, making it efficient for large-scale and distributed computation.[5] **L-BFGS**: A quasi-Newton optimization algorithm that approximates the Hessian matrix using past gradient information, leading to faster convergence compared to SGD. Suitable for problems where higher accuracy and fewer iterations are required.[5]	**Use case:** A large health insurance company wants to predict healthcare costs per patient to adjust insurance premiums and reimbursement rates. The dataset consists of 50 million patient records, including Patient demographics (age, gender, location, medical history), Hospital visits & procedures, Chronic conditions (e.g., diabetes, hypertension), Total healthcare costs (dependent variable, highly skewed) Since healthcare costs are continuous, non-negative, and right-skewed, we use Gamma regression (a type of GLM) with a log link function. **Reasons:** SGD in Spark can process fresh patient cost data as it arrives, allowing for real-time cost prediction, unlike Base R and Python statsmodels.GLM() must re-train from scratch whenever new data is added. Base R glm() may require

			thousands of IRLS iterations, rendering it inefficient for millions of records, whereas Spark R's L-BFGS optimization requires fewer iterations to converge, resulting in faster calculation time. Hence, Spark optimization is regarded preferable in this instance.
f.	Scikit-learn	**Stochastic Gradient Descent(SGD):** Stochastic Gradient Descent is an optimization method used for fitting linear models, including GLMs, by minimizing a loss function iteratively. In the context of GLMs, it can optimize for different types of losses (e.g., log loss for logistic regression, squared hinge for SVM). SGD is especially useful for large datasets due to its efficiency.[6] **Coordinate descent:** Coordinate Descent is an optimization algorithm that optimizes one parameter at a time while keeping the others fixed. It is used in models such as Logistic Regression with L1 regularization (Lasso). It is more efficient in high-dimensional settings when only a subset of the features are expected to be relevant.[6] **Newton's method:** Newton's method is an optimization technique that uses the second-order derivative (Hessian matrix) to find the optimal parameters. Newton-CG is a variant that uses conjugate gradients for solving the problem efficiently in higher dimensions. It is particularly useful for GLMs with a logistic regression or Poisson regression setup.[6] **LBFGS:** LBFGS is an optimization algorithm that is well-suited for large-scale machine learning problems. It approximates Newton's method and is more memory-efficient. It is often used for GLMs and can handle both L1 and L2 regularization.[6] **Gradient descent(GD):** For GLMs like Ridge or Lasso	**Use case:** A pharmaceutical company wants to predict the sales volume of a new drug based on several factors such as the Price of the drug, Advertising spend, Sales channel (online, retail, wholesale), Seasonality (cold and flu season), Promotions and discounts. The target variable (number of drug units sold) is a count variable, which is best modeled using Poisson regression. This model assumes that the target follows a Poisson distribution and is suitable for count-based data like drug sales volume. **Reasons:** Scikit-learn works well with other machine learning tools like cross-validation, hyperparameter tuning, and pipelines. This makes it easier to automate and optimize the model-building process, but Base R lacks easy integration with machine learning pipelines or hyperparameter tweaking tools such as grid search, making scaling models more difficult. Scikit-learn provides easy access to cross-validation and model selection, allowing us to fine-tune the model and choose the optimal configuration. In contrast, Base R lacks these model tuning tools and requires extra packages or manual implementation for tasks such as cross-validation. Thus, Scikit-learn outperforms in this scenario.

		regression, gradient descent can be used to optimize the model parameters with the goal of minimizing a cost function, while also applying regularization (L1 or L2). This helps avoid overfitting by penalizing large coefficients.[6] **Active set method:** The Active Set Method is another optimization algorithm that is used for models like Logistic Regression with L1 regularization. It solves the problem by iteratively selecting a set of active coefficients that are most relevant for optimization.[6]	

References:

1. glm function - RDocumentation. www.rdocumentation.org. https://www.rdocumentation.org/packages/stats/versions/3.6.2/topics/glm
2. Eddelbuettel D. CRAN Task View: High-Performance and Parallel Computing with R. *R-projectorg*. Published online January 15, 2025. doi:https://cran.r-project.org/view=HighPerformanceComputing
3. Generalized Linear Models — dask-ml 2024.4.5 documentation. Dask.org. Published 2024. Accessed March 28, 2025. https://ml.dask.org/glm.html
4. Generalized Linear Models — spark.glm. Apache.org. Published 2025. Accessed March 28, 2025. https://spark.apache.org/docs/3.5.0/api/R/reference/spark.glm.html
5. spark/docs/mllib-optimization.md at master · apache/spark. GitHub. Published 2025. Accessed March 28, 2025. https://github.com/apache/spark/blob/master/docs/mllib-optimization.md
6. 1.1. Linear Models — scikit-learn 0.24.0 documentation. scikit-learn.org. https://scikit-learn.org/stable/modules/linear_model.html